Graphics Hardware

Why do you hate XML?

```xml
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//
<plist version="1.0">
<dict>
  <!-- array of downloads. -->
  <key>items</key>
  <array>
   <dict>
    <!-- an array of assets to download -->
     <key>assets</key>
      <array>
        <dict>
         <!-- Required. The asset kind. -->
          <key>kind</key>
          <string>software-package</string>
          <!-- Optional. md5 is used here for chu
          <key>md5-size</key>
          <integer>10485760</integer>
          <!-- Array of md5 hashes for each "md5-
          <key>md5s</key>
          <array>
            <string>41fa64bb7a7cae5a46bfb45821ac8
            <string>51fa64bb7a7cae5a46bfb45821ac8
            <string>61fa64bb7a7cae5a46bfb45821ac8
          </array>
          <!-- required. the URL of the package t
          <key>url</key>
          <string>https://www.theacmeinc.com/apps
        </dict>
      </array>
    </dict>
  </array>
</dict>
```

```json
{
  "items": [
    {
      "assets": [
        {
          "kind": "software-package",
          "md5-size": 10485760,
          "md5s": [
            "41fa64bb7a7cae5a46bfb45821ac8b99",
            "51fa64bb7a7cae5a46bfb45821ac8b98",
            "61fa64bb7a7cae5a46bfb45821ac8b97"
          ],
          "url": "https://www.theacmeinc.com/apps/myapp.pkg"
        }
      ]
    }
  ]
}
```

According to GPT-3.5, at any rate...

Now imagine that mess, but spread out over hundreds of keys/elements.

XML is fine at what it does (provide a low-level hierarchical data storage format which is somewhat human-readable) but that doesn't mean I have to like looking at the stuff.

How do you save objects into files?

- saveTable
- saveJSONObject
- saveJSONArray

Is Friday the last day with a hands-on?

It will be the last day with a graded hands-on.

The bonus classes may or may not come with hands-on materials as a way to explore the material, but they will not be for credit.

Why do most web APIs use JSON as the data transfer format?

Two reasons:

- HTTP is a text transfer protocol, so we need to transfer the bulk of our data in text format.
- JavaScript is the de-facto standard of the web, and most JavaScript implementations natively understand JSON.

If you look at remote requests that don't have to go through HTTP, you often see different formats preferred (e.g. gRPC packs data using Protobuf instead of JSON).

Universality is a powerful reason in favor of JSON in many applications.

Can you make graphs or plots in Processing?

Absolutely! In fact, you could make an entire plotting library or program if you want to.

🤔 But why would you want to? 🤔

Asking the right questions, I see.

You probably wouldn't---existing tools in Python and R are going to be miles better. But CSV import doesn't just have to be for data visualization. For example, you can store game maps in CSV, user options in JSON, etc.

Can you include audio in Processing?

Absolutely. In fact, in the regular-semester version of this class, sound is a requirement for the final project (there's just not enough time to cover it in summer).

https://processing.org/reference/libraries/sound/index.html

Can you turn a Processing app into a standalone app?

File > Export Application

Yahtzee Croshaw (of Zero Punctuation fame) did a series where he developed a complete game every month for 12 months.

This episode has great advice on how to make satisfying animations for games (2:41-4:46).

Warning: contains harsh language (as do all his videos).

https://www.youtube.com/watch?v=QgC-RssfTYA&list=PLFERI12uu5pltdQ40tPXm9j1ENoAG3wgH&index=2

Computer Hardware

Typical Programming Language

Very high-level ideas: variables, collections, loops, functions, etc.

```
1 for (TableRow r : t.rows()) {
2   String id = r.getString("object");
3   float x = r.getFloat("x");
4   float y = r.getFloat("y");
5 }
```

Things like ArrayLists, functions, and even loops do not exist on the level of the CPU!

CPU Programming Language

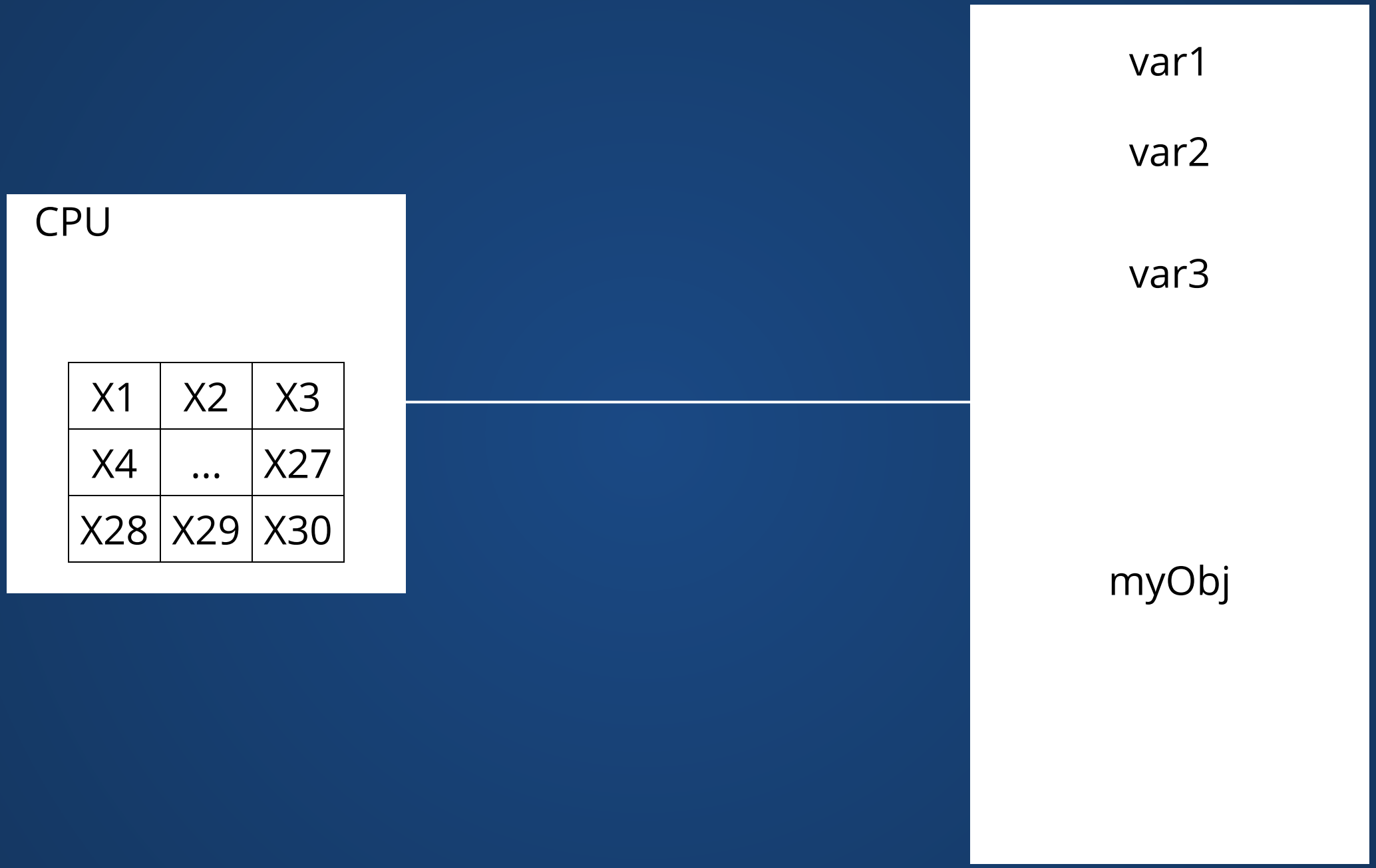
An *incredibly* barebones programming language suitable for running directly on a CPU.

Limited number of variables (in our case, we'll call them X1 through X30).

Commands to load from and store to main memory in case our local variables aren't enough.

Commands to manipulate variables, e.g. add, subtract, etc.

Commands to jump around.


CPU
X1
X2
X3
X4
...
X27
X28
X29
X30
var1
var2
var3
myObj

Assembly Code (x64)

```
push   %rbp
mov    %rsp,%rbp
mov    %edi,-0x4(%rbp)
mov    -0x4(%rbp),%eax
imul   %eax,%eax
pop    %rbp
ret


push   %rbp
mov    %rsp,%rbp
mov    $0x5,%edi
call   1d <main+0xe>
mov    %eax,%esi
lea    0x0(%rip),%rax        # 26 <main+0x17>
mov    %rax,%rdi
mov    $0x0,%eax
call   33 <main+0x24>
mov    $0x0,%eax
pop    %rbp
ret
```

Example Language

Arithmetic Commands

- ADD
- SUB
- MUL
- DIV

Load/Store Commands

- LOAD
- STORE

Control Command

- BRANCH
- CALL

```
1  X1 = ADD X1 X2
2  X2 = SUB X2 X3
3  X3 = MUL X3 X4
4  X4 = DIV X4 X5
5
6  X5 = LOAD [myObj]
7  STORE X5 TO [myObj]
8
9  BRANCH (X4 == 0) 4
10 X7 = CALL square(X2)
```

Rule: once we finish executing one instruction, we go to the one below it, unless we execute a branch/call.

Example Compilation

```
1  x = 5;
2  x++;
```

```
1  X1 = ADD 0 5
2  X1 = ADD X1 1
3  STORE X1 TO [x]
```

Example Compilation

```
1 if (x > 5){
2   x += 1;
3 } else {
4   x += 2;
5 }
6 println(x);
```

```
1 X1 = LOAD [x]
2 BRANCH (X1 > 5) 5
3 X1 = ADD X1 2
4 BRANCH (TRUE) 6
5 X1 = ADD X1 1
6 STORE X1 to [x]
7 CALL println(X1)
```

Example Compilation

```
1  for(int i = 0; i < 10; i++){
2    x += 5;
3  }
```

```
1  X1 = LOAD [x]
2  X2 = ADD 0 0 // X2 = i
3  X1 = ADD X1 5
4  X2 = ADD X2 1
5  BRANCH (X2 < 10) 3
6  STORE X1 to [x]
```

What does this do?

```
1  X1 = LOAD [number]
2  X2 = ADD 0 0
3  BRANCH (X1 <= 0) 8
4  X1 = SUB X1 2
5  X2 = ADD X2 1
6  BRANCH (X1 > 0) 4
7
8  CALL print(X2)
```

Branches Slow Down Computers

Branch-Heavy Instructions

Branch-heavy workloads make it harder to do things quickly!

Real-World Example

In a month, you are going to Alaska for one week and to Hawaii for a week afterwards.

In a month, you are going to either Alaska or Hawaii for two weeks, but we're not going to tell you which one until the day before.

What strategies could you use to deal with the second scenario?

Dependent Workloads

Dependent Instructions also make it hard to do things quickly!

You and five friends all get together to help someone restore some furniture. Which can be done faster?

One piece of furniture which needs to be sanded, primed, and stained.

Three pieces of furniture, one which needs to be sanded, one which needs to be primed, and one which needs to be stained.

Typical Workloads

Remember that both if-else and loops get compiled into branches.

How many branches do you think your typical code has?

Studies have shown that in typical programs, as many as 1/6 instructions are branches!

A very simple data-dependent branch chain!

```
1  while (x < 50){
2    x += 5;
3  }
```

```
1  X1 = LOAD [x]
2  X1 = ADD X1 5
3  BRANCH (X1 < 50) 2
4  STORE X1 to [x]
```

CPUs

Because most workloads have so many branches and data dependencies, modern CPUs dedicate a lot of their engineering to these sorts of problems:

- **Branch predictors**: try to guess which branch will be taken
- **Register renaming**: if some data is dependent, can rename certain variables.
- **Out-of-Order Execution:** Try to execute instructions out of order, then put the results back together in a way that looks like this never happened.

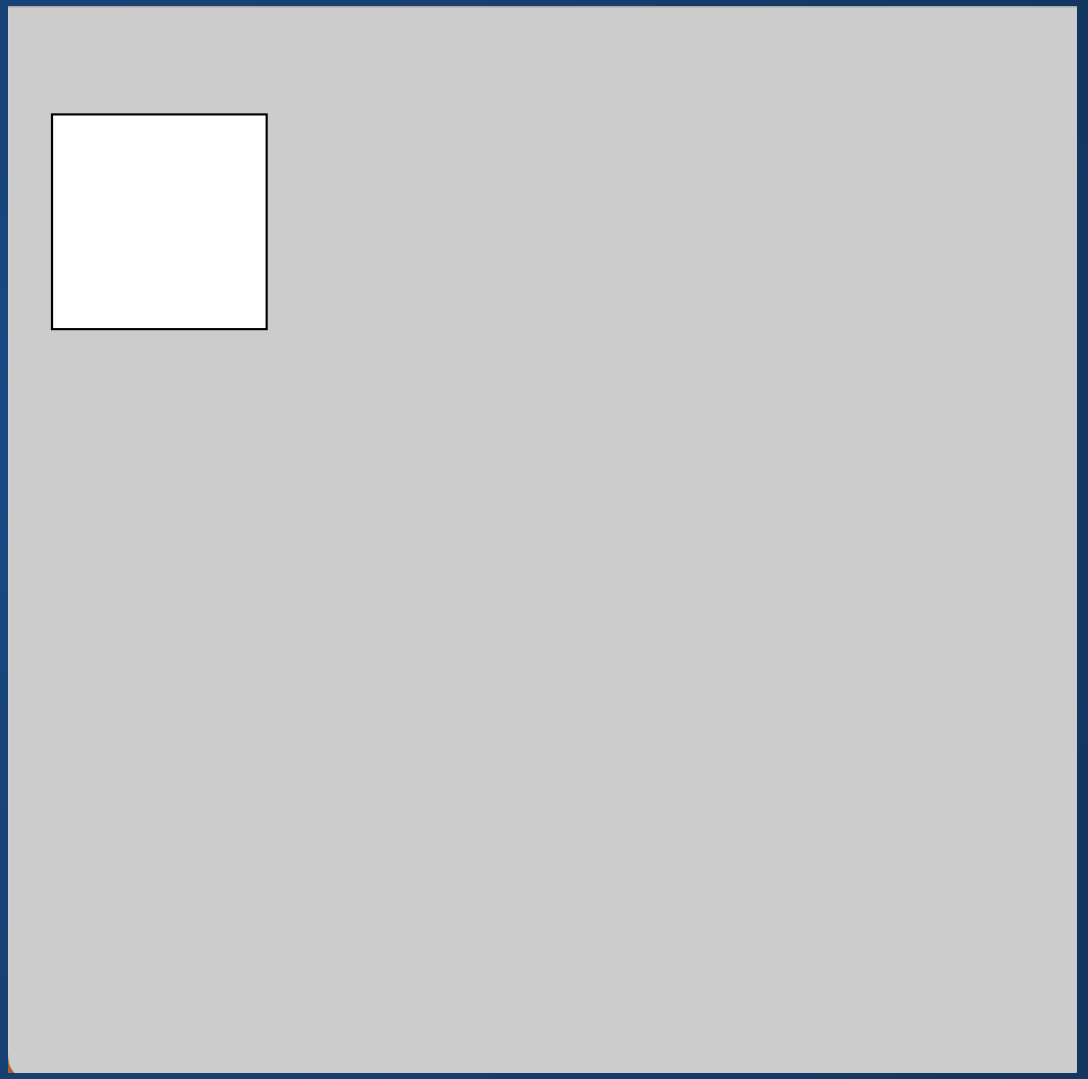
This necessarily makes it slower (or not as fast) to execute highly independent, branchless code.

Graphics Workloads

Let's Take a Look Back....

Drawing Shapes

```
1  void setup(){
2    size(500, 500);
3  }
4
5  void draw(){
6    rect(20,50,100,100);
7  }
```



Processing Images

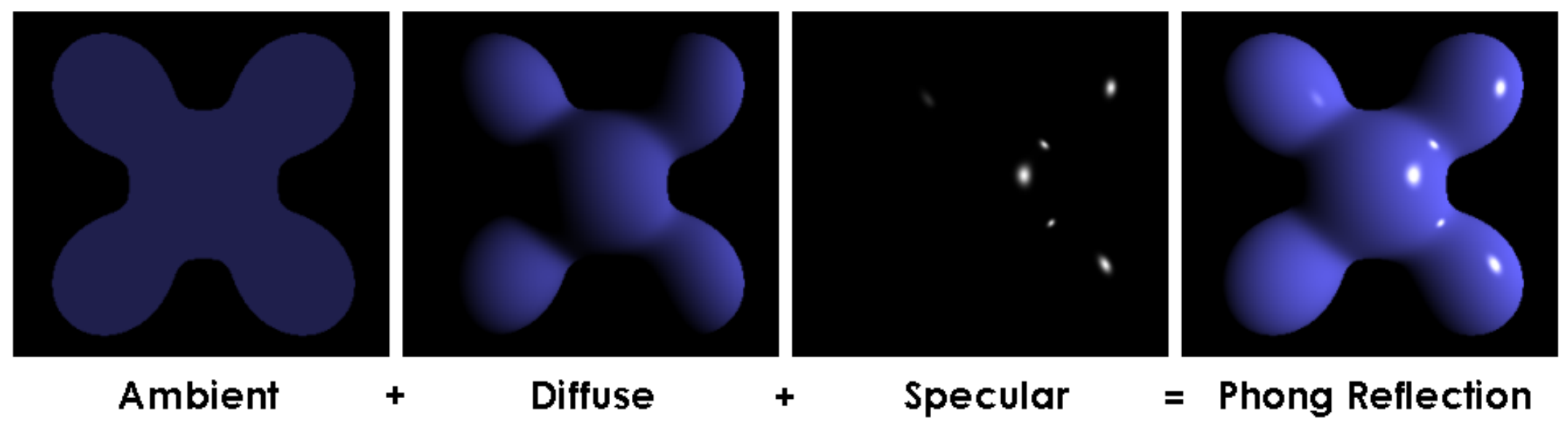
Apply this kernel to each pixel in the source image to get the result.

-1	0	1
-2	0	2
-1	0	1

Any data dependencies?

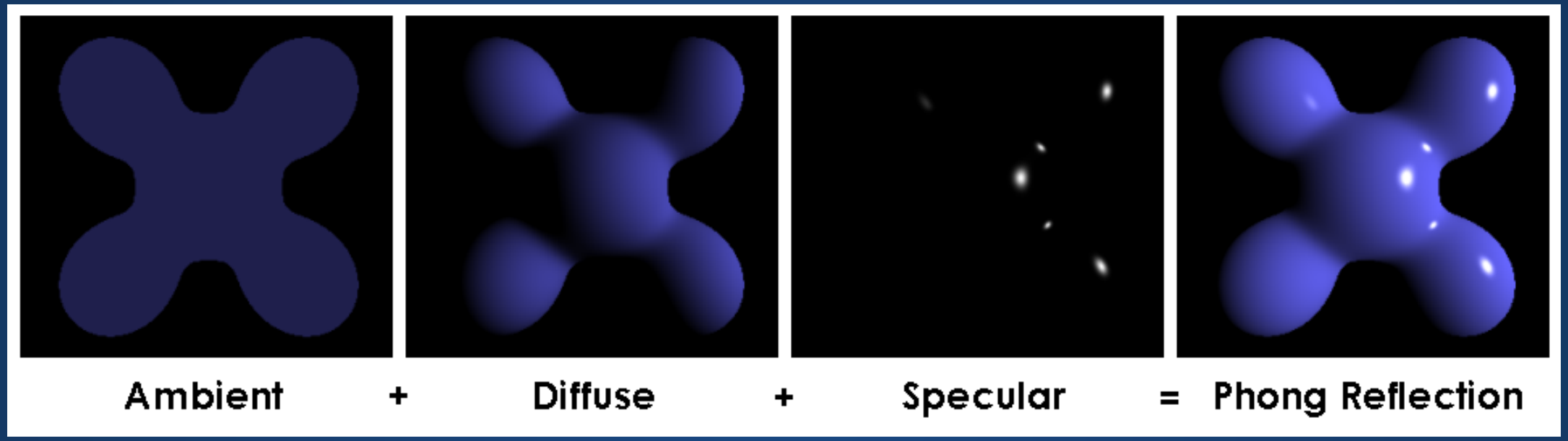
Any branches?

Drawing in 3D



- Ambient depends on just ambient coefficient
- Diffuse value at a pixel depends on the surface normal of the shape at that pixel, and the direction of the light.
- Specular value at a pixel depends on surface normal, direction of light, and direction of camera.

Drawing in 3D



No data dependencies! (Color of a pixel does not directly affect the color of its neighbors)

No branches!

Transforms

Recall that transforms are performed as matrix-vector or matrix-matrix multiplications.

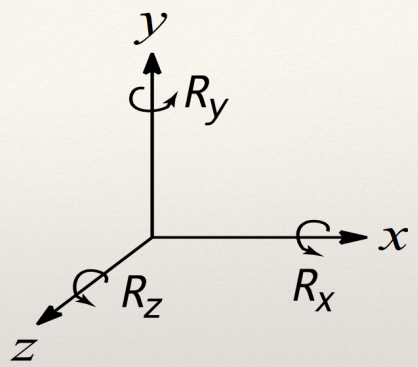
Each matrix-multiply is branch-free and dependence free.

What about transforming 1000 points?

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(Use right hand rule)

Graphics Workloads are different!

Unlike most CPU workloads, workloads in graphics tend to be

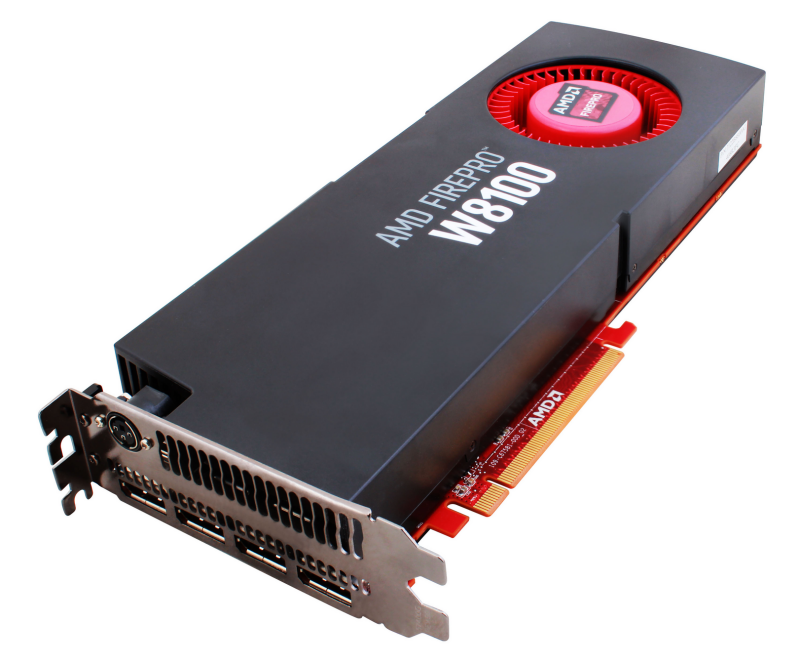
- Low-branch
- Data-independent
- Parallel

but involve large amounts of data movement.

We can exploit this by using special hardware which specializes in this kind of workload to speed up graphics!

Graphics Cards

Graphics Processing Unit



GPUs are processing units which specialize in the types of computations needed for graphics.

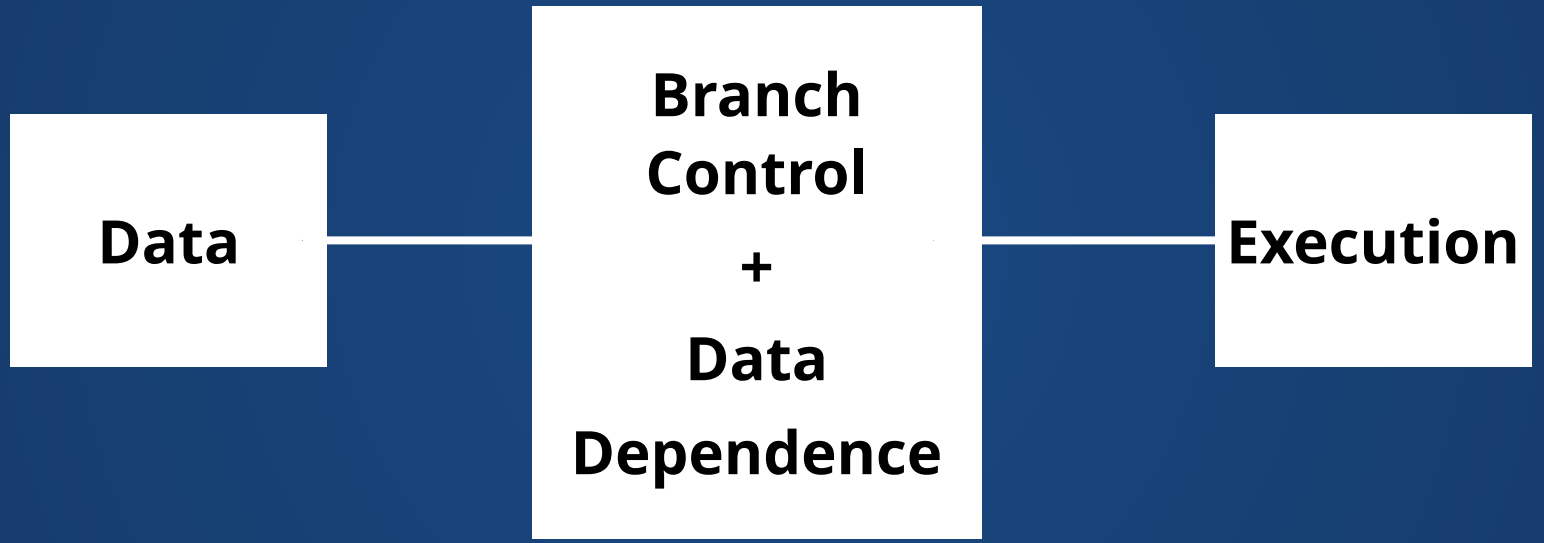
Sacrifice the ability to deal effectively (or at all!) with branches and data dependence.

In exchange, gain the ability to move massive amounts of data and execute on it very quickly (as long as you don't have branches or data dependence!)

Typical GPU: ~10,000 independent operations per cycle.

Typical CPU: ~100 independent operations per cycle.

CPU



GPU

How do we control a GPU?

Writing GPU Programs

In the same way we don't want to write x64 assembly for CPUs, we don't really want to write in the GPU-specific instruction set for GPUs.

But the kinds of programs that GPUs are good at running (or are even capable of running!) look very different from CPU programs.

How do we write programs for GPUs while respecting these differences?

Libraries and Shaders

Write programs for the GPU in a special *shading language*. These languages are good at expressing the kinds of programs that GPUs run well.

Need some way to get the program + data to the GPU. This is usually handled by the operating system.

Need a program that takes advantage of this: this is written by the programmer (you!) or by some higher-level program which generates shaders (details next time).

```
1 #version 330 core
2 out vec4 FragColor;
3
4 in vec4 vertexColor;
5
6 void main()
7 {
8     FragColor = vertexColor;
9 }
```

Hands-On: GL Speed Test

```
1  void setup(){
2    size(1200, 800);
3  }
4
5  void draw(){
6     int NUM_CIRCLES = 500;
7     int R = 100;
8     for(int i = 0; i < NUM_CIRCLES; i++){
9       int ix = int(random(0, 1200));
10      int iy = int(random(0, 800));
11      ellipse(ix, iy, R, R);
12    }
13    println(frameRate);
14 }
```

This code attempts to draw `NUM_CIRCLES` circles every frame. If `NUM_CIRCLES` is too high, Processing will not be able to keep up, and the frame rate will drop.

Hands-On: GL Speed Test

1. Experiment by changing `NUM_CIRCLES` until your the `frameRate` that Processing reports is about 30fps. Note: `frameRate` will not be accurate until about a second has passed. Record this number.

2. Change the size call to `size(1200, 800, P2D);` which enables a GPU-accelerated renderer if your computer supports it.

3. Run the code under the `P2D` renderer and record the approximate `frameRate` you get.

4. Change `NUM_CIRCLES` until you once again get about 30fps. Record this number.

5. Submit your file with the numbers you recorded in steps 1, 3, and 4 in a comment.

Index Cards!

1. Your name and EID.

2. One thing that you learned from class today. You are allowed to say "nothing" if you didn't learn anything.

3. One question you have about something covered in class today. You *may not* respond "nothing".

4. (Optional) Any other comments/questions/thoughts about today's class.